Expertise
and insight
for the future

Duc Minh Luong Nguyen

Detect COVID-19 from Chest X-Ray images using Deep Learning

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

May 2020

Author Title Number of Pages Date	Duc Minh Luong Nguyen Detect COVID-19 from Chest X-Ray images using Deep Learning 36 pages 25 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department (ICT)

The COVID-19 pandemic has been causing devastating impacts on the well-being of people around the world as well as the global economy. Motivated by the effort of the open source community on collecting the COVID-19 dataset and the success of Deep Learning on previous studies with chest radiography, this thesis builds a Deep Convolutional Neural Network in order to detect COVID-19 using only chest X-Ray images. This project uses modern Deep Learning techniques such as using pretrained networks and fine-tuning as well as regularizations such as data augmentation and dropout to fight overfitting. The resulting model achieves an overall accuracy of 93% on the most realistic task of detecting COVID-19 patients among healthy normal people despite being trained on a dataset of only 115 images for each class. Out of 100 patients who do have COVID-19, the model accurately identifies 96 patients and misses out 4 patients. Out of 100 normal patients who do not have COVID-19, the model accurately identifies 91 patients as healthy and misclassifies 9 patients as COVID-19 positive. Overall, the model did a decent job as a COVID-19 detector but still has limitations and is far from being production-ready.

Keywords	Deep Learning, Neural Network, CNN, AI, COVID-19

Contents

1 Introduction

The COVID-19 pandemic has been causing devastating impacts on the well-being of people around the world as well as the global economy. As of the 24th of May 2020, more than 5 million people are confirmed to have been infected with the virus and more than 300 thousand people have died worldwide. In Finland, COVID-19 has infected more than 6500 people and caused more than 300 deaths. One effective method to combat COVID-19 is to increase the testing capacity. However, this is not possible in some places such as New York City due to the lack of testing kits. Motivated by the effort of the open source community on collecting the COVID-19 dataset and the success of Deep Learning on previous studies with chest radiography, this thesis builds a Deep Convolutional Neural Network in order to detect COVID-19 using only chest X-Ray images.

This thesis is organized as follows. First, section 2 briefly summarizes the history of Deep Learning. Section 3 gives a quick introduction to Convolutional Neural Networks. Section 4 discusses the main reasons for the rapid advancement of Deep Learning in recent years. Section 5 is about preparing the data including setting up the working environment, collecting the dataset, and preprocessing data. Section 6 presents the implementation of the models and the training process. Section 7 discusses the results of the models. Finally, section 8 draws conclusions and discusses the limitations and suggestions for future improvements.

2 Brief history of Deep Learning

Deep Learning, a subset of Artificial Intelligence, is a Machine Learning technique which can enable computers to solve problems that were otherwise unable to explicitly program them to do. Even though the principled method to train deep networks was available since the 1980s, it still was not able to scale to large networks and Neural Networks research fell into a dark period.

In 2006, Hinton GE et al. [1] showed that deep neural networks could possibly be trained effectively by achieving a state-of-the-art result on the MNIST handwritten digit dataset. This paper was a breakthrough that rekindled and brought back beliefs in Deep Learning.

In 2012, Krizhevsky A et al. [2] won the annual ImageNet Large Scale Visual Recognition Challenge by a huge margin with the first and only Deep Learning based system. This marked the introduction of modern Deep Learning, which since then was developed and used in various kinds of applications.

Nowadays, Deep Learning is used everywhere, from spam filters to advertisements recommendation systems, from face recognition to self-driving cars, from predicting stock prices to generating music.

3 Convolutional Neural Networks

An artificial neuron is generally a function that takes in vectors as inputs, performs non-linear transformations on them then finally outputs a value. A Deep Neural Network is built by stacking these neurons together hierarchically when each neuron is connected to every neuron in the previous layer. This architecture works well with tabular data but is not suitable to deal with images in computer vision problems. For example, if the network has an input layer of size 1000x1000x3, which is only about one-megapixel image, followed by just one 1000-unit layer, the number of parameters would be three billion already. This approach is not scalable because when image size becomes bigger, the computational cost and memory requirements will explode.

Convolutional Neural Network (CNN) is one of the most popular types of Deep Neural Networks that is very useful for computer vision tasks. CNNs take images as input, filter them using convolution operations to get a final vector that summarizes interesting features of an image. That vector will then be fed into a series of Fully Connected Layers to do classification on it. A layer of ten 3x3x3 filters only has 280 parameters and this number will stay the same even if the input image size increases, which makes training deeper and larger networks possible. Figure 1 shows the architecture of AlexNet, the first CNN that won the ImageNet competition in 2012. It achieved 15.3% error rates [2, 7], while the runner-up only achieved 26.2%.

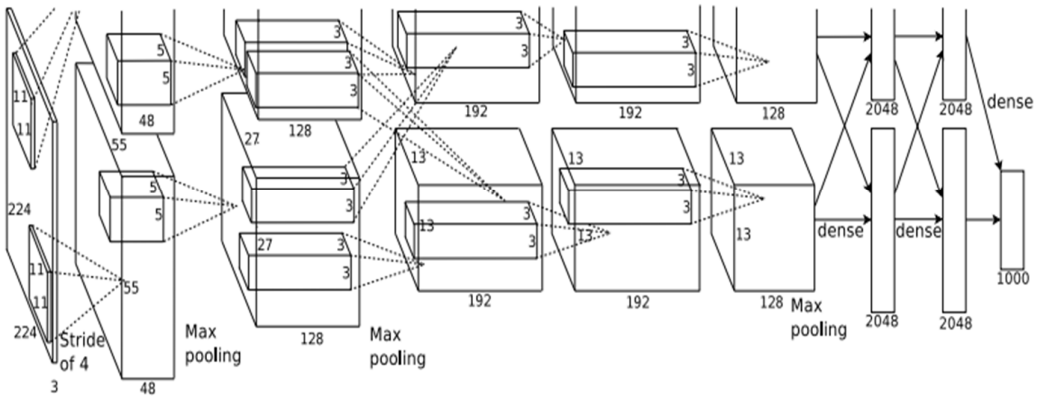

Figure 1: Architecture of AlexNet. Copied from Krizhevsky et al. (2012) [2, 5]

As shown in Figure 1, AlexNet consists of eight layers, five convolutional layers followed by three fully connected layers. Going deeper into the network, the height and width decrease, while the number of channels (depth) increases. This architecture of a series of convolutional layers, followed by pooling layers and finally a few fully connected layers became a common pattern of CNNs. AlexNet also used ReLU instead of Sigmoid as activation function and two GPUs which made training faster [2, 3]. Since then, CNNs have been widely used in all kinds of computer vision problems such as object detection, semantic segmentation, face recognition, self-driving cars.

4. Why did Deep Learning take off?

There are four main reasons why deep learning took off.

First, it is easier to collect data thanks to an immense number of sensors and the digitalization of society when people spend a lot of time online. The traditional Machine Learning methods cannot deal with larger datasets, whereas Deep Learning methods will keep getting better results with more data. [4, 20].

Second, the advancement in computer hardware, especially Graphics processing units (GPUs), helped reduce the training time dramatically. After having an idea for a network, researchers can implement and train the network, then fine-tune the model based on previous results. With the training time reduced from hours to minutes, this process becomes highly iterative.

Third, improvements in the software also contributed to the rise of Deep Learning. Better algorithmic techniques such as switching from Sigmoid to Rectified Linear Units as activation function have made training Deep Neural Networks faster [2, 3]. Libraries such as Tensorflow and Pytorch have also lowered the entry level of Deep Learning so that not only researchers but students and practitioners can apply it to solve their problems.

Finally, following deep learning success in computer vision in 2013 is a wave of industry investment from startups trying to join the deep learning hype as well as tech giants. In 2014, DeepMind, an AI startup, was acquired by Google for more than $500 million. In the Alphabet earnings call 2015, Google CEO Sundar Pichai stated to apply machine learning across all their products. [6,22]. This wave of investment has resulted in an increasing number of Deep Learning jobs.

5 Dataset

5.1 Overview of the Dataset

The first and most important part of a Deep Learning project is collecting data. For this project, chest X-ray images are needed from 4 classes: COVID-19, normal, pneumonia, and COVID-19 negative. Currently, there is no available dataset that includes images from all those 4 classes. As a result, a new dataset is collected by combining the Kaggle Chest X-ray dataset [6] with the COVID19 Chest X-ray dataset collected by Dr. Joseph Paul Cohen of the University of Montreal [3]. Both datasets consist of posterior anterior chest images of patients which is the most preferred and common type of chest X-ray images.

Dr. Joseph Paul Cohen, Postdoctoral Fellow at the University of Montreal, recently published a public open dataset containing chest X-ray and CT images of patients suffering from COVID-19 as well as MERS, SARS, and ARDS [3]. As the COVID19 dataset is being updated daily as more cases are published, the images used in this project is the instance available on the 18th of March 2020.

The dataset from Kaggle is provided by the Radiological Society of North America (RSNA). In August 2018, the RSNA hosted a $30.000 prize money competition in Kaggle challenging the machine learning community to build an algorithm to detect pneumonia from chest radiographs [6]. The specific requirement of this competition is that the model should locate the lung opacities, which is an object detection problem and is more challenging than image classification in this project. However, this dataset still offers thousands of images of normal and pneumonia infected patients, which are valuable for this project.

To generate a dataset that is ready for training, images from those two datasets must be downloaded first. Next, the desired images are selected and saved to the correctly labeled folder. Images are then loaded and preprocessed by being transformed into numpy arrays with desired size in order to be ready to feed into the training process.

5.2 Preparing data

5.2.1 Setting up Google Colaboratory

Before getting the data, it is necessary to set up the working environment for the project. Google Colaboratory (colab) is a free cloud-based with no setup required Jupyter notebook environment. Colaboratory allows users to write and execute code and access powerful computing resources for free from the browser. Most importantly colab generously provides GPU which helps significantly speed up the training process which is computing intensive. For these reasons, colab has become very popular among Deep Learning and Data Science enthusiasts who might not necessarily own a PC with expensive GPUs.

However, a small problem occurs in how to save and load data in colab. The dataset could be downloaded and saved directly to colab's disk but will disappear after the runtime resets. To persist data and avoid running the download code again and again, colab needs to connect with google drive where the data is saved. Data is processed and saved to the dataset folder in google drive. In order to do that, on the left side panel of colab, on the Files section there is an option for Mount Drive.

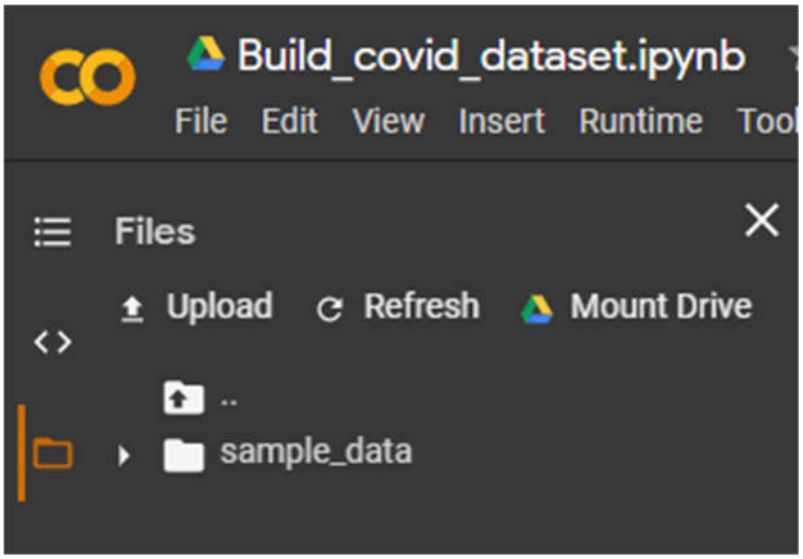

Figure 2 Mount drive

Selecting Mount Drive, a dialog will show up asking for permission to access Google Drive.

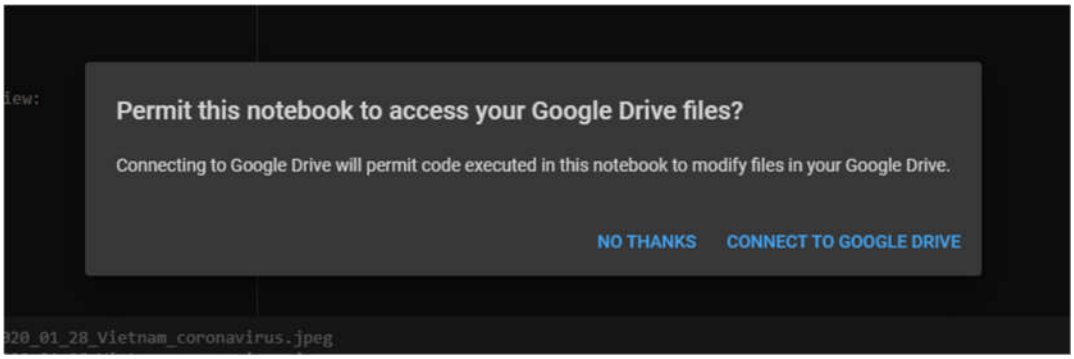

Figure 3 Confirm dialog to mount drive

Now colab has connected to google drive and can directly access files from google drive.

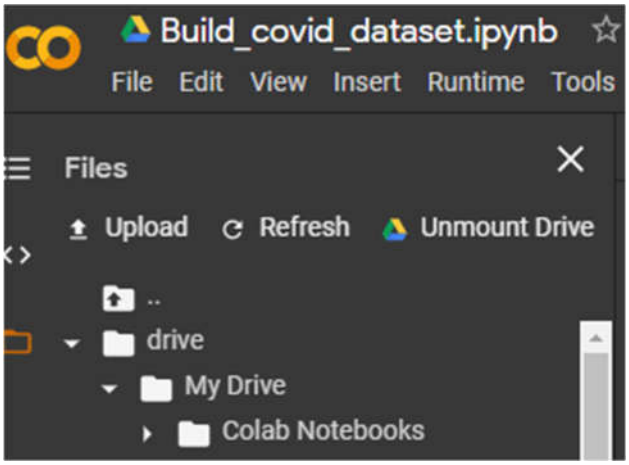

Figure 4 Drive is mounted

One more problem before actual coding is the default folder name of the mounted google drive is /drive/My Drive/ which causes errors when trying to access the file using that path. A simple workaround is to create a symbolic link to get a cleaner path.

```
1  # how to rename the default name 'My Drive'?
2  # https://support.google.com/drive/thread/8914333?hl=er
3  !ln -s /content/drive/My\ Drive/covid19 /myDrive
```

Figure 5 Create a symbolic link

From now on, the path to any files inside the `/content/drive/My\ Drive/covid19` folder would become `/myDrive` instead.

5.2.2 Getting COVID-19 chest X-ray images

The first step in building the dataset is to download the COVID chest X-ray dataset by cloning Dr. Cohen's Github repository [6]. The `images` folder contains CT and chest X-ray images with different chest views from patients of COVID-19 as well as other illnesses such as MERS, SARS, and ARDS. The `metadata.csv` file must be parsed into a pandas data frame in order to filter and select only COVID-19 positive chest X-ray images with posterior anterior (PA) view, the most preferred and common view, from the `images` folder.

```python
import pandas as pd
import shutil
import os

# Selecting all combination of 'COVID-19' patients with 'PA' X-Ray view
virus = "COVID-19" # Virus to look for
x_ray_view = "PA" # View of X-Ray

covidPath = '/myDrive/covid-chestxray-dataset'
csv = os.path.join(folderPath, 'metadata.csv')  # Meta info
imageDir =  os.path.join(folderPath, 'images') # Directory of images

outputDir = '/myDrive/dataset/covid' # Output directory to store selected images

covid_df = pd.read_csv(csv)

count = 0 # number of COVID-19 PA images

# loop over the rows of the COVID-19 data frame
for (i, row) in covid_df.iterrows():
  if row["finding"] != virus or row["view"] != x_ray_view:
    continue
  filename = row["filename"].split(os.path.sep)[-1]
  inputPath = os.path.sep.join([imageDir, filename])
  outputPath = os.path.sep.join([outputDir, filename])
  shutil.copy2(inputPath, outputDir)
  count += 1
```

Figure 6 Select COVID-19 chest PA images

The code section in Figure 6 is based on the `select_covid_patient_X_ray_images.py` file from Dr. Cohen's repository. The paths need to be reconfigured and a counter is added to quickly get the number of suitable images. There was also a bug from that provided file which was surprisingly merged into the repository. In the final line, `shutil.copy2(imageDir, outputPath)`, the first parsed argument is supposed to be a path to a file, not a directory. Currently there is an opened pull request that fixed this bug. Running the code block, 115 Covid-19 positive images is saved to the `/dataset/covid` located on google drive and ready to be accessed later in colab. The number of collected images is significantly higher than other earlier researches since the dataset is still continuously updated.

```python
def get_images_from_folder(imgDir):
  images = []
  # https://stackoverflow.com/questions/25868109/python-read-all-files-in-directory-and-subdirectories
  for root, directories, filenames in os.walk(imgDir):
    for filename in filenames:
      if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        file = os.path.join(root,filename)
        images.append(np.array(image.load_img((file), target_size=(224, 224))))
  random.shuffle(images)
  return images
```

Figure 7 Get images from folder function

This function takes in a path to the image directory as an argument, iterates through all the files, selects all the images files, and transforms them into a numpy array of size 224x224. Next, the images array is randomly shuffled and finally returned.

```python
imgDir_covid = '/myDrive/dataset/covid'
images_covid = get_images_from_folder(imgDir_covid)

f = plt.figure(figsize=(10,10))
plt.title('COVID-19 positive', fontsize=16)
for i in range(4):
  sp = f.add_subplot(2, 2, i+1)
  sp.axis('Off')
  plt.imshow(images_covid[i], cmap=plt.cm.binary)
```

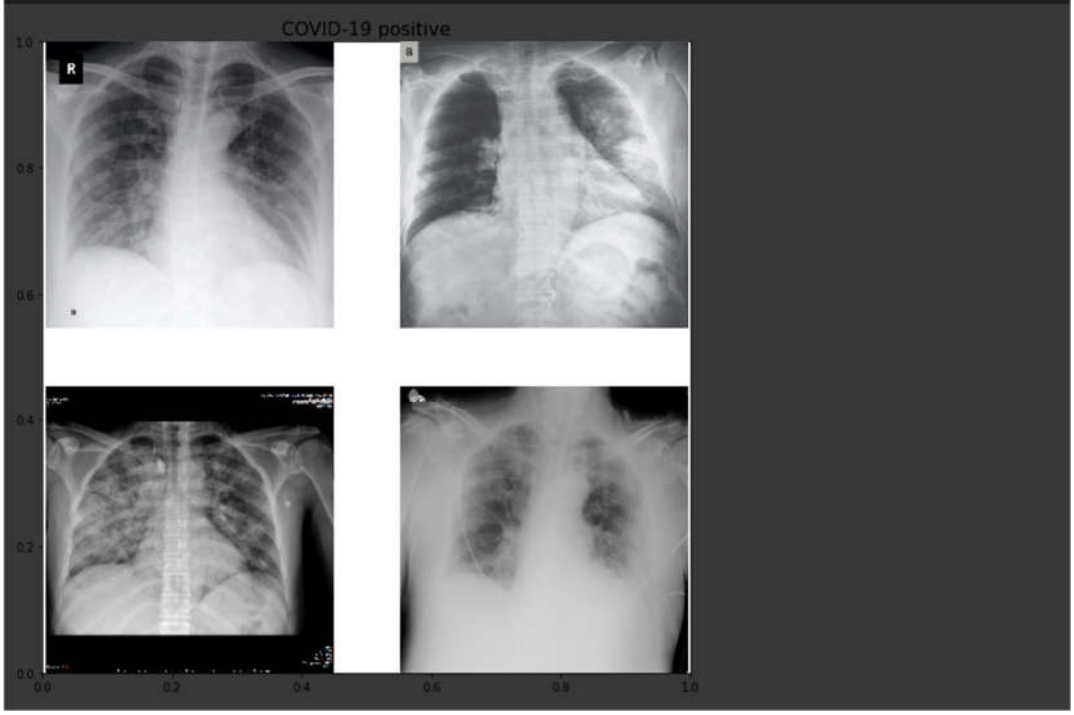

Figure 8 COVID-19 chest X-Ray images

Image 8 shows examples of chest X-ray images of COVID-19 infected patients.

5.2.3 Getting Pneumonia chest X-ray images

The next step is to collect a dataset of chest X-ray images of patients with pneumonia as well as healthy ones from Kaggle. One way to do this is to manually download the dataset, unzip it, and save the images to the project folder in google drive. Kaggle also offers a public API for downloading their dataset using the command line [8], which is the approach used in this project. First, `pip install kaggle` is called so that the kaggle command can be called using the command line. Next, an API token needs to be generated from the 'Account' tab by login into your kaggle account. Finally, before being able to download the dataset, the user needs to sign up for the RSNA pneumonia detection challenge.

```
1  !kaggle competitions download -c rsna-pneumonia-detection-challenge

Downloading rsna-pneumonia-detection-challenge.zip to /content
100% 3.66G/3.66G [00:49<00:00, 78.2MB/s]
100% 3.66G/3.66G [00:49<00:00, 79.0MB/s]
```

Figure 9 Download the RSNA pneumonia dataset

The 3.66 Gb dataset is downloaded directly and is accessible from google colab in less than 1 minute.

```
1  ! unzip -q -n 'rsna-pneumonia-detection-challenge.zip' -d '/myDrive'
2
```

Figure 10 Extract the dataset

Next, the dataset .zip file is unzipped to the project folder in google drive. The dataset consists of train and test images folder as well as .csv files such as the one in the figure 11 below which specify the classes of those images. Patients that were classified as No Lung Opacity / Not Normal were not diagnosed with pneumonia.

patientId	class
0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
00569f44-917d-4c86-a842-81832af98c30	No Lung Opacity / Not Normal
006cec2e-6ce2-4549-bffa-eadfcd1e9970	No Lung Opacity / Not Normal
00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity
00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity

Show 10 per page 1 2 10 100 1000 2000 3000 3020 3023

Figure 11 Content of the .csv file

The unzipped images are in the /stage_2_train_images folder. The 'stage_2_train_labels.csv' file indicates whether a patient is normal or has pneumonia.

```
1  kaggle_datapath = '/myDrive'
2  kaggle_csvname = 'stage_2_detailed_class_info.csv' # get all the normal from here
3  kaggle_csvname2 = 'stage_2_train_labels.csv' # get all the 1s from here since 1 indicate pneumonia
4  kaggle_imgpath = 'stage_2_train_images'
5
6  outputDir = '/myDrive/dataset/pneumonia' # Output directory to store selected images
7
8  non_covid_df = pd.read_csv(os.path.join(kaggle_datapath, kaggle_csvname2), nrows=None)
9
10 count = 0
11 patients = set()
12
13 # loop over the rows of the non-covid data frame
14 for (i, row) in non_covid_df.iterrows():
15   if int(row["Target"]) != 1:
16     continue
17   patient = row['patientId']
18   if patient in patients:
19     continue
20   dcmImage = dicom.dcmread(os.path.join(kaggle_datapath, kaggle_imgpath, patient + '.dcm'))
21   pixel_array_numpy = dcmImage.pixel_array
22   imgName = patient + '.png'
23   cv2.imwrite(os.path.join(outputDir, imgName), pixel_array_numpy)
24
25   count += 1
26   patients.add(patient)
27   print(imgName, count)
28   if (count == 115):
29     break
```

```
00436515-870c-4b36-a041-de91049b9ab4.png 1
00704310-78a8-4b38-8475-49f4573b2dbb.png 2
00aecb01-a116-45a2-956c-08d2fa55433f.png 3
00c0b293-48e7-4e16-ac76-9269ba535a62.png 4
00f08de1-517e-4652-a04f-d1dc9ee48593.png 5
0100515c-5204-4f31-98e0-f35e4b00004a.png 6
010ccb9f-6d46-4380-af11-84f87397a1b8.png 7
```

Figure 12 Script to collect pneumonia images

Line 8 reads the .csv file into a pandas data frame. Inside the folder, the image file name is in the format of patientid.dcm. The loop goes through the rows in the data frame and only selects the patient with the target value 1 which indicates the patient has pneumonia). Since from the figure 11 above there are multiple images from a single patient, a

set data structure is used to make it easier to check whether an image from a patient has already been collected. The .dcm images are converted into numpy arrays which are then saved as .png files in the output folder. A counter is used to make sure that only 115 images would be collected, the same number as the COVID-19 positive images. Figure 13 below shows the examples of chest X-ray images from patients with pneumonia.

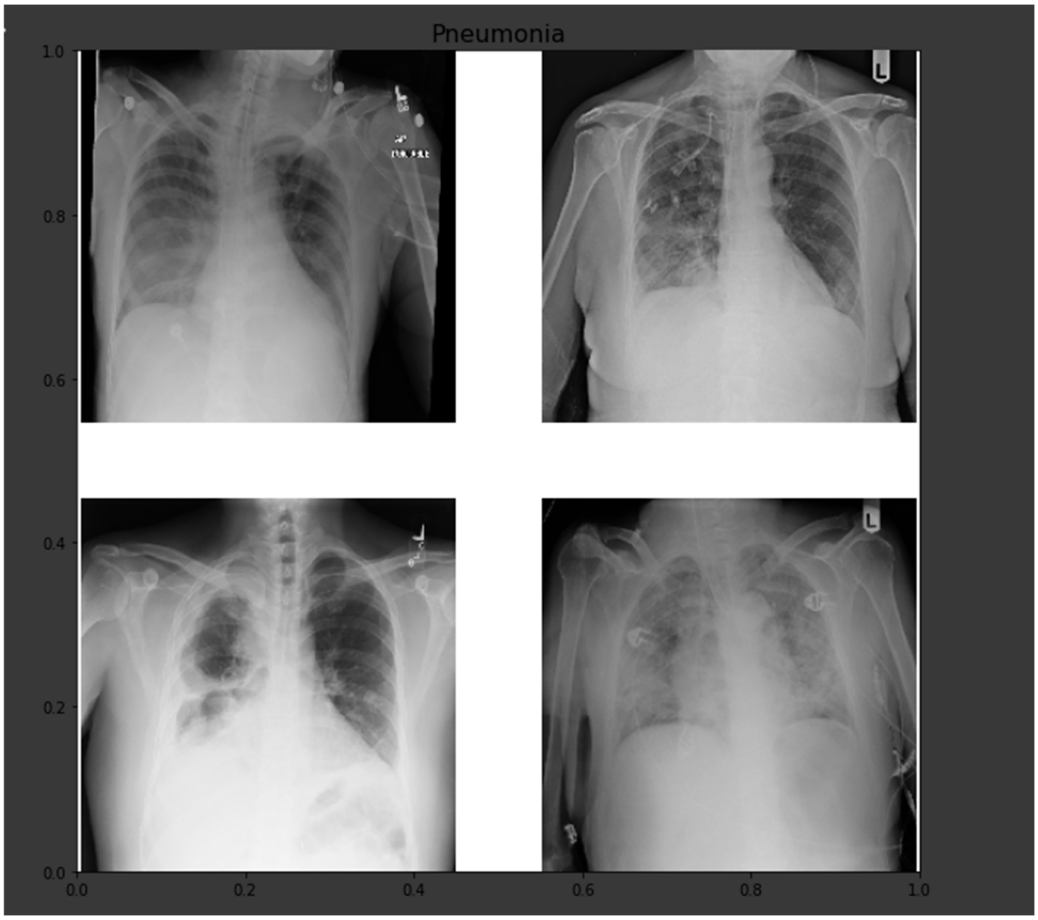

Figure 13 Examples of chest X-ray images of patients with pneumonia

From the examples of chest X-ray images of patients with pneumonia, the differences between the black lungs and the white areas below are not clear. The reason is that for patients with pneumonia, the air in their lungs is replaced by fluids, bacteria, etc. which makes these areas grey and fuzzy. [9].

5.2.4 Getting normal chest X-ray images

The chest X-ray images of people without pneumonia are in the /stage_2_train_images folder alongside with the ones with pneumonia. The 'stage_2_detailed_class_info.csv' file indicates whether a patient is normal or has pneumonia.

```
1  kaggle_datapath = '/myDrive'
2  kaggle_csvname = 'stage_2_detailed_class_info.csv' # get all the normal from here
3  kaggle_imgpath = 'stage_2_train_images'
4
5  outputDir = '/myDrive/dataset/normal' # Output directory to store selected images
6
7  non_covid_df = pd.read_csv(os.path.join(kaggle_datapath, kaggle_csvname), nrows=None)
8
9  count = 0
10 patients = set()
11
12 # loop over the rows of the non-covid data frame
13 for (i, row) in non_covid_df.iterrows():
14   if row["class"] != 'Normal':
15     continue
16   patient = row['patientId']
17   if patient in patients:
18     continue
19   dcmImage = dicom.dcmread(os.path.join(kaggle_datapath, kaggle_imgpath, patient + '.dcm'))
20   pixel_array_numpy = dcmImage.pixel_array
21   imgName = patient + '.png'
22   cv2.imwrite(os.path.join(outputDir, imgName), pixel_array_numpy)
23
24   count += 1
25   patients.add(patient)
26   print(imgName, count)
27   if (count == 115):
28     break
```

```
068ae297-8397-463e-927e-cb6693aeb8b5.png 58
068e882a-df1e-4d62-92a0-9eef12024b01.png 59
06941d8e-2034-4503-b8d5-0eb13fc52f75.png 60
069554dc-f5b0-4947-be4c-873dd6360b5a.png 61
069cfd47-0169-43e7-89a1-0be0fa24105b.png 62
06a7820e-8f38-4958-b871-518030a63432.png 63
06b2f933-3ea2-4477-ac27-18f732d1f4e1.png 64
```

Figure 14 Script to collect chest X-ray images of healthy patients

Line 7 reads the .csv file into a pandas data frame. Inside the folder, the image file name is at the format of patientid.dcm. The loop from line 13 goes through the rows in data frame and only selects the patient with the value of "Normal" from the "class" column which indicates the patient does not have pneumonia). Since there are multiple images from a single patient, a set data structure is used to make it easier to check whether an image from a patient has already been collected in line 17. In line 20, the .dcm images are converted into numpy arrays which are then saved as .png files in the output folder in line 22. A counter is used to make sure that only 115 images would be collected, the same number as the COVID-19 positive images.

Figure 15 below shows the examples of chest X-ray images from patients without pneumonia.

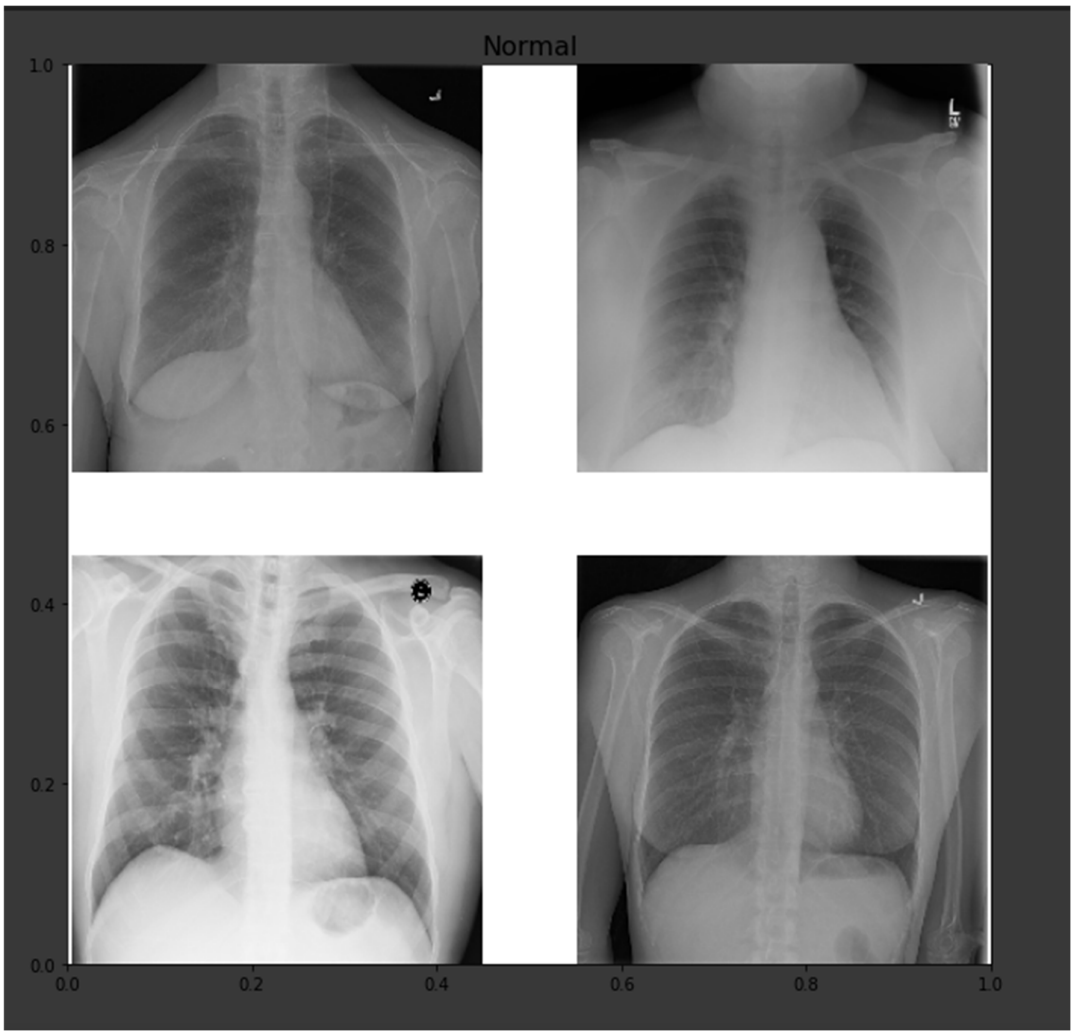

Figure 15 Examples of chest X-ray images of healthy patients

From the examples of chest X-ray images of patients without pneumonia, there is clearer separation between the black lungs and the white areas below compared to ones from patients with pneumonia.

5.2.5 Getting COVID-19 negative chest X-ray images

The chest X-ray images of people with and without pneumonia are in the /stage_2_train_images folder. The NonCOVID images folder is generated by combining 57 images of patients without pneumonia and 58 images with pneumonia. The script is just the combination of the previous 2 scripts with the same steps. First, the .csv files are read into pandas data frames (lines 8, 9) which are used to determine whether the patient is pneumonia positive or not (lines 21, 38). The images are then converted to .png files and finally saved to the desired output folder using the `convert_dcm_to_png` function (line 11).

```
1  kaggle_datapath = '/myDrive'
2  kaggle_csvname = 'stage_2_detailed_class_info.csv' # get all the normal from here
3  kaggle_csvname2 = 'stage_2_train_labels.csv' # get all the 1s from here since 1 indicate pneumonia
4  kaggle_imgpath = 'stage_2_train_images'
5
6  outputDir = '/myDrive/dataset/noncovid' # Output directory to store selected images
7
8  non_covid_df1 = pd.read_csv(os.path.join(kaggle_datapath, kaggle_csvname), nrows=None)
9  non_covid_df2 = pd.read_csv(os.path.join(kaggle_datapath, kaggle_csvname2), nrows=None)
10
11 def convert_dcm_to_png(patient):
12   dcmImage = dicom.dcmread(os.path.join(kaggle_datapath, kaggle_imgpath, patient + '.dcm'))
13   pixel_array_numpy = dcmImage.pixel_array
14   imgName = patient + '.png'
15   cv2.imwrite(os.path.join(outputDir, imgName), pixel_array_numpy)
16
17 count = 0
18 patients = set()
19 # loop over the rows of the non-covid data frame 1
20 for (i, row) in non_covid_df1.iterrows():
21   if row["class"] != 'Normal':
22     continue
23
24   patient = row['patientId']
25   if patient in patients:
26     continue
27   convert_dcm_to_png(patient)
28
29   count += 1
30   print(patient, count)
31   patients.add(patient)
32   if (count == 57):
33     break
34
35 count = 0
36 # loop over the rows of the non-covid data frame 2
37 for (i, row) in non_covid_df2.iterrows():
38   if int(row["Target"]) != 1:
39     continue
40
41   patient = row['patientId']
42   if patient in patients:
43     continue
44   convert_dcm_to_png(patient)
45
46   count += 1
47   print(patient, count)
48   patients.add(patient)
49   if (count == 58):
50     break

003d8fa0-6bf1-40ed-b54c-ac657f8495c5 1
009482dc-3db5-48d4-8580-5c89c4f01334 2
009eb222-eabc-4150-8121-d5a6d06b8ebf 3
```

Figure 16 Script to collect chest X-ray images of COVID-19 negative patients

Figure 17 below shows the examples of chest X-ray images from patients with and without COVID-19.

```
images = [images_covid[0], images_pneumonia[0], images_normal[0], images_covid[1],images_pneumonia[1],  images_normal[1]]
labels = ['COVID-19', 'Pneumonia', 'Normal']

f = plt.figure(figsize=(20,10))

for i in range(6):
  sp = f.add_subplot(2, 3, i+1)
  sp.axis('Off')
  sp.set_title(labels[i%3], fontsize=16)
  plt.imshow(images[i], cmap=plt.cm.binary)
```

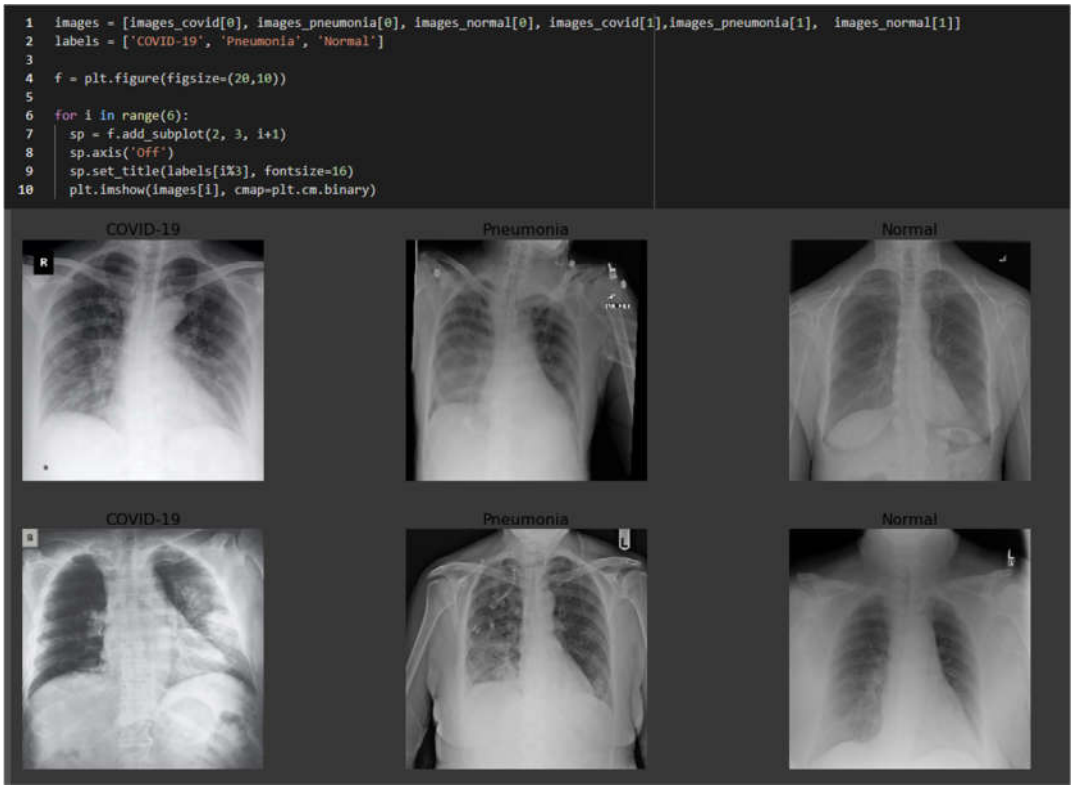

Figure 17 Examples of chest X-ray images of patients with and without COVID-19

With the images of the 3 classes COVID-19 positive, pneumonia and normal side by side, the differences between healthy and not healthy lungs are obvious. There are visible opacities at the lower boundary of the lungs on the images from patients with COVID-19 and pneumonia which indicates the lung tissues in that area are not healthy. The task to detect COVID-19 patients among healthy people would be easier than among patients with not healthy lungs for any reason.

5.3 Preprocessing data

The next step before feeding the data into the neural networks is to preprocess the data. This includes reading the images as numpy arrays, resizing the images to the desired size, normalizing the images (making the pixel value between 0 and 1), encoding the labels, splitting the dataset into training and validating data and finally data augmentation.

For this project, only 115 images have been collected for each class with the bottleneck coming from the number of available images of patients with COVID-19. The main concern with this limited amount of data is that the model will just remember all the training images which would result in its poor generalization ability, making it perform badly against validation images that it has not seen before.

Data augmentation is a technique that could help fight overfitting. Basically, during the training phase, the generator will randomly perform some transformations on the original image such as rotating, shearing, shifting, flipping. As a result, the model will not be fed the same image twice so that it cannot just remember all the images from the training set. The model trained with data augmentation is more robust and can generalize better.

Implementing it in keras is very simple with `ImageDataGenerator` in line 6. Figure 18 shows an example of performing augmentation on a randomly selected image and saving the result images in the “/preview” folder.

```
6  datagen = ImageDataGenerator(
7    rotation_range=40,
8    width_shift_range=0.2,
9    height_shift_range=0.2,
10   shear_range=0.2,
11   zoom_range=0.2,
12   horizontal_flip=True,
13   fill_mode='nearest')
14
15
16 dataPath = '/myDrive/dataset/covid'
17 imagePaths = list(paths.list_images(dataPath))
18
19 img = load_img(random.choice(imagePaths))  # this is a PIL image
20 x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
21 x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)
22
23 # the .flow() command below generates batches of randomly transformed images
24 # and saves the results to the `preview/` directory
25 i = 0
26 for batch in datagen.flow(x, batch_size=1, save_to_dir='/myDrive/preview', save_prefix='covid', save_format='jpeg'):
27   i += 1
28   if i > 20:
29     break  # otherwise the generator would loop indefinitely
```

Figure 18 Example script of data augmentation

Figure 19 shows how the data augmentation strategy above looks like. All these 20 images are generated from a single image.

```
1  imgDir_covid = '/myDrive/preview'
2  images_covid = get_images_from_folder(imgDir_covid)
3
4  f = plt.figure(figsize=(15,15))
5
6  for i in range(20):
7    sp = f.add_subplot(4, 5, i+1)
8    sp.axis('Off')
9    plt.imshow(images_covid[i], cmap=plt.cm.binary)
```

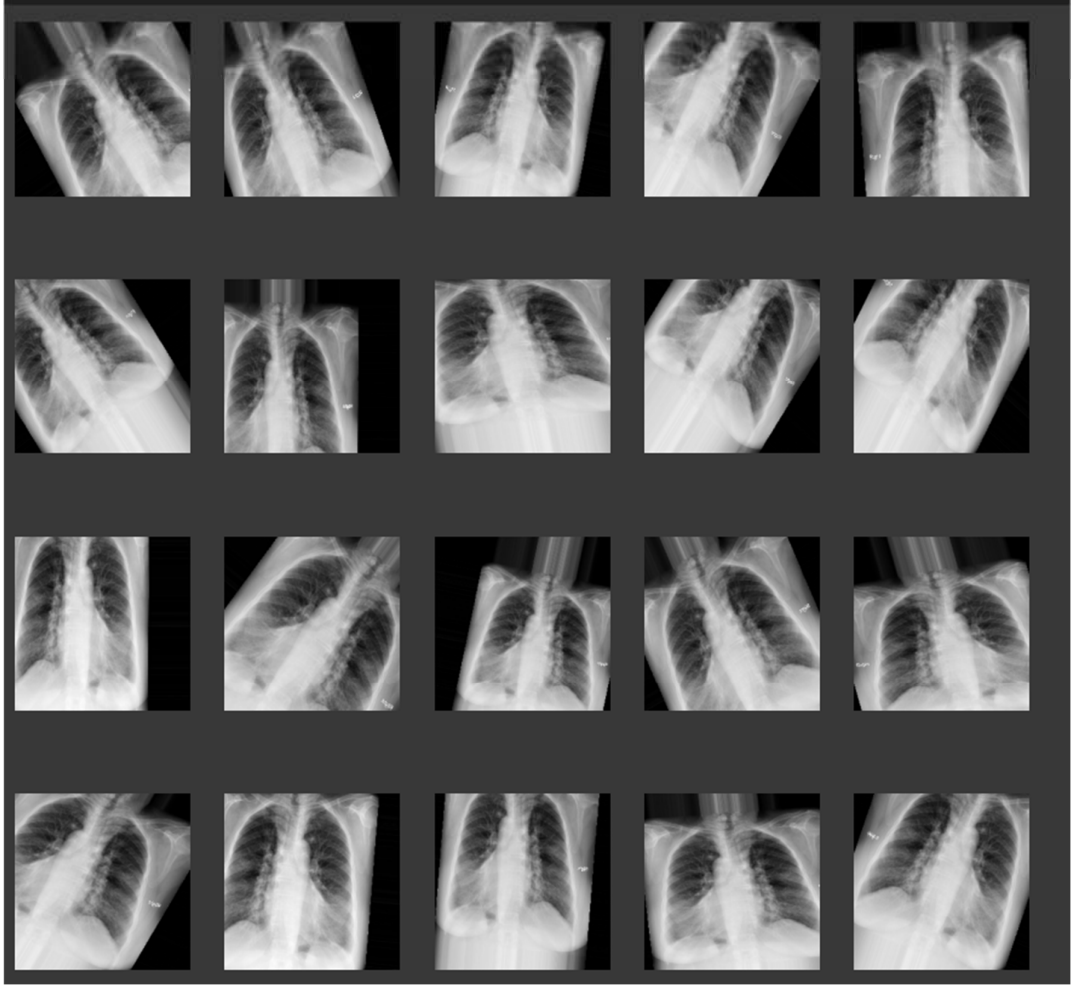

Figure 19 Data augmentation examples

Figure 20 shows the process of loading and preprocessing X-Ray images data. Line 9 generates a list of all the paths to the images from the dataset. For each path, the label is extracted according to the current subproject (COVID vs non-COVID, COVID vs pneumonia, COVID vs normal). Lines 22 to 24 load the image, reorder the order of channels

to RGB and resize it to be ready for the neural networks. The data and labels lists are updated on lines 27-28. Data and labels lists are transformed into numpy arrays while the image's pixel values are normalized to range between 0 and 1. Next, the labels array is one-hot encoded which means the label will be 0 or 1 instead of for example covid or non-covid. Line 42 splits the images into a training and testing dataset. Finally, a simpler data augmentation generator is initialized.

```
7  dataPath = '/myDrive/dataset'
8
9  imagePaths = list(paths.list_images(dataPath))
10 data = []
11 labels = []
12
13 # loop over the image paths
14 for imagePath in imagePaths:
15   # extract the class label from the filename
16   label = imagePath.split(os.path.sep)[-2]
17   if (label not in ['covid', 'noncovid']):
18     continue
19
20   # load the image, swap color channels, and resize it to be a fixed
21   # 224x224 pixels while ignoring aspect ratio
22   image = cv2.imread(imagePath)
23   image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
24   image = cv2.resize(image, (224, 224))
25
26   # update the data and labels lists, respectively
27   data.append(image)
28   labels.append(label)
29
30 # convert the data and labels to NumPy arrays while scaling the pixel
31 # intensities to the range [0, 255]
32 data = np.array(data) / 255.0
33 labels = np.array(labels)
34 print(labels[:10])
35 # perform one-hot encoding on the labels
36 lb = LabelBinarizer()
37 labels = lb.fit_transform(labels)
38 print(labels[:10])
39
40 # partition the data into training and testing splits using 80% of
41 # the data for training and the remaining 20% for testing
42 (trainX, testX, trainY, testY) = train_test_split(data, labels,
43   test_size=0.20, stratify=labels, random_state=42)
44
45 # initialize the training data augmentation object
46 trainAug = ImageDataGenerator(
47   rotation_range=15,
48   fill_mode="nearest")
```

Figure 20 Preprocess data script

After these preprocessing steps above, the generator can be consumed by the model method `fit_generator` from keras.

6 Training models

6.1 Training a Convolutional Neural Network from scratch

For an image classification task, a convolutional neural network is the default tool. Training a simple convnet from scratch is a decent initial baseline of each sub-project. Figure 21 shows the implementation of a standard convnet.

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer='rmsprop',
  metrics=["accuracy"])

model.summary()
```

Figure 21 Implementation of a convnet

The network consists of three 3x3 convolution layers using ReLU activation function followed by 2D max-pooling layers and two fully connected layers on top of them. Max-pooling layers are used to reduce the size of the representation which makes computation faster. Dropout is a regularization method to help reduce overfitting by randomly ignoring a certain number of each layer's outputs (50% in this project), which as a result makes the model more robust. The model ends with a single unit with sigmoid activation which is the default for binary classification problems.

Figure 22 shows the summary of the architecture of our simple convnet. The total number of trainable parameters of this network is already nearly 2.8 million.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
activation_15 (Activation)   (None, 222, 222, 32)      0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 109, 109, 32)      9248
_________________________________________________________________
activation_16 (Activation)   (None, 109, 109, 32)      0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 52, 52, 64)        18496
_________________________________________________________________
activation_17 (Activation)   (None, 52, 52, 64)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 26, 26, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                2768960
_________________________________________________________________
activation_18 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_19 (Activation)   (None, 1)                 0
=================================================================
Total params: 2,797,665
Trainable params: 2,797,665
Non-trainable params: 0
_________________________________________________________________
```

Figure 22 Network summary

The model is now ready to train. Thanks to Colab's free GPU, training only took less than 3 seconds an epoch.

```
H = model.fit(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

```
Epoch 1/20
23/23 [==============================] - 2s 94ms/step - loss: 0.7177 - accuracy: 0.6576 - val_loss: 0.5188 - val_accuracy: 0.6957
Epoch 2/20
23/23 [==============================] - 2s 94ms/step - loss: 0.6673 - accuracy: 0.6848 - val_loss: 0.5828 - val_accuracy: 0.7609
Epoch 3/20
23/23 [==============================] - 2s 94ms/step - loss: 0.6331 - accuracy: 0.6739 - val_loss: 0.5335 - val_accuracy: 0.8043
Epoch 4/20
23/23 [==============================] - 2s 92ms/step - loss: 0.6004 - accuracy: 0.7337 - val_loss: 0.5078 - val_accuracy: 0.7391
Epoch 5/20
23/23 [==============================] - 2s 92ms/step - loss: 0.5641 - accuracy: 0.7120 - val_loss: 0.4566 - val_accuracy: 0.8261
Epoch 6/20
23/23 [==============================] - 2s 93ms/step - loss: 0.5848 - accuracy: 0.7446 - val_loss: 0.4377 - val_accuracy: 0.7826
Epoch 7/20
23/23 [==============================] - 2s 94ms/step - loss: 0.5773 - accuracy: 0.7337 - val_loss: 0.4306 - val_accuracy: 0.8043
Epoch 8/20
23/23 [==============================] - 2s 95ms/step - loss: 0.5114 - accuracy: 0.7500 - val_loss: 0.3594 - val_accuracy: 0.8261
Epoch 9/20
23/23 [==============================] - 2s 92ms/step - loss: 0.4583 - accuracy: 0.7935 - val_loss: 0.4259 - val_accuracy: 0.7826
Epoch 10/20
23/23 [==============================] - 2s 93ms/step - loss: 0.4043 - accuracy: 0.8261 - val_loss: 0.4052 - val_accuracy: 0.7609
Epoch 11/20
23/23 [==============================] - 2s 92ms/step - loss: 0.4708 - accuracy: 0.7826 - val_loss: 0.4170 - val_accuracy: 0.8261
Epoch 12/20
23/23 [==============================] - 2s 93ms/step - loss: 0.3770 - accuracy: 0.8478 - val_loss: 0.5225 - val_accuracy: 0.8261
Epoch 13/20
23/23 [==============================] - 2s 93ms/step - loss: 0.4777 - accuracy: 0.8152 - val_loss: 0.3547 - val_accuracy: 0.7826
Epoch 14/20
23/23 [==============================] - 2s 92ms/step - loss: 0.3938 - accuracy: 0.8261 - val_loss: 0.4409 - val_accuracy: 0.8261
Epoch 15/20
23/23 [==============================] - 2s 94ms/step - loss: 0.3831 - accuracy: 0.8261 - val_loss: 0.4092 - val_accuracy: 0.8261
Epoch 16/20
23/23 [==============================] - 2s 95ms/step - loss: 0.3732 - accuracy: 0.8478 - val_loss: 0.3608 - val_accuracy: 0.8478
Epoch 17/20
23/23 [==============================] - 2s 95ms/step - loss: 0.3162 - accuracy: 0.8859 - val_loss: 0.3278 - val_accuracy: 0.8696
Epoch 18/20
23/23 [==============================] - 2s 94ms/step - loss: 0.3229 - accuracy: 0.8478 - val_loss: 0.5026 - val_accuracy: 0.7391
Epoch 19/20
23/23 [==============================] - 2s 94ms/step - loss: 0.3214 - accuracy: 0.8859 - val_loss: 0.4404 - val_accuracy: 0.8478
Epoch 20/20
23/23 [==============================] - 2s 95ms/step - loss: 0.2426 - accuracy: 0.9076 - val_loss: 0.3748 - val_accuracy: 0.8478
```

Figure 23 COVID-19 vs normal training process

```
N = EPOCHS
plt.style.use("ggplot")
plt.figure(figsize=(15,15))

plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 vs Normal")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

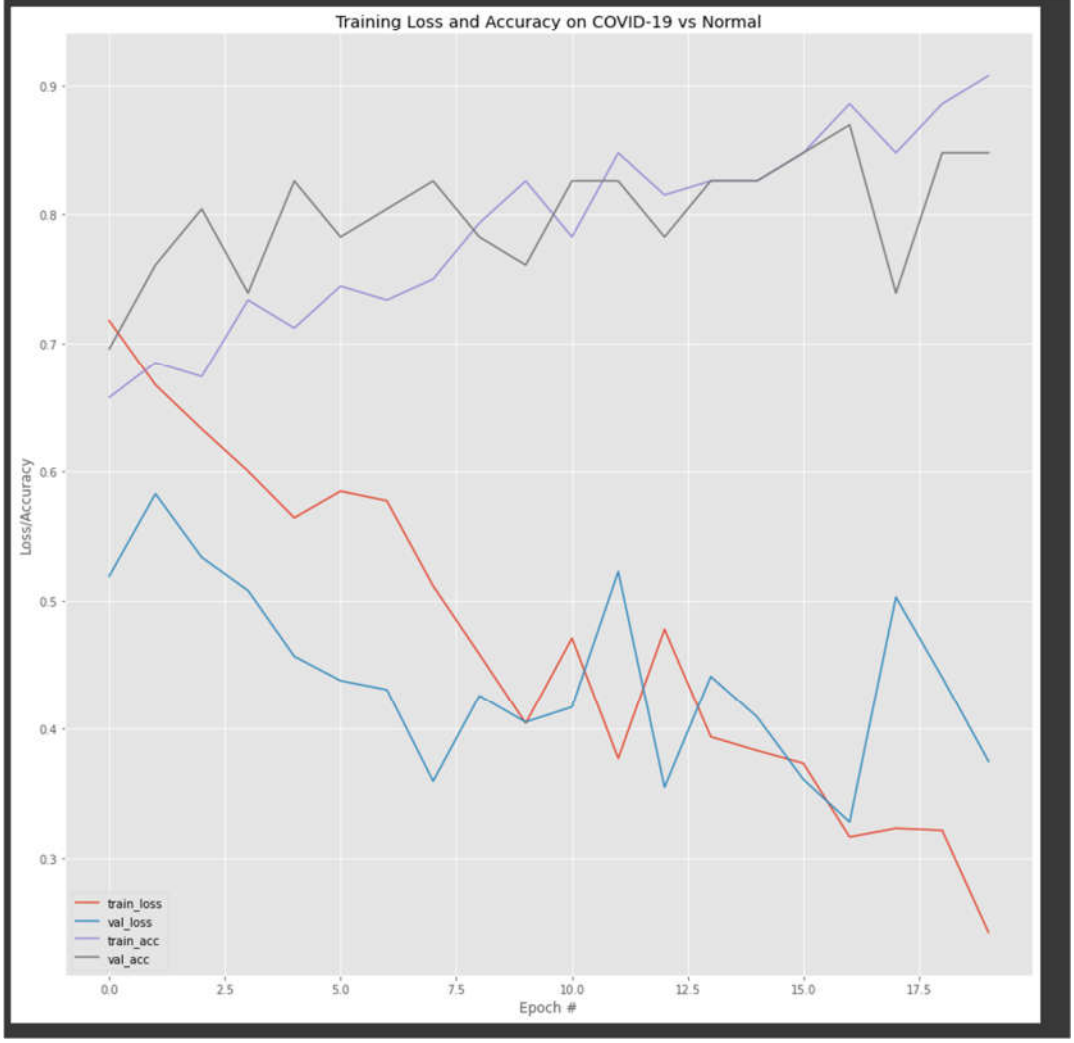

Figure 24 COVID-19 vs normal training accuracy/lost history

After 20 epochs, the model reaches a validation accuracy of 85% of classifying COVID-19 positive patients and normal people. Figure 24 shows that the model does not suffer badly from overfitting since there is no big divergence between training and validation loss and accuracy. It is likely that continuing the training process with a higher number of epochs would result in even higher validation accuracy.

```
Epoch 1/20
23/23 [==============================] - 2s 84ms/step - loss: 1.3542 - accuracy: 0.5163 - val_loss: 0.6757 - val_accuracy: 0.7391
Epoch 2/20
23/23 [==============================] - 2s 77ms/step - loss: 0.6890 - accuracy: 0.5924 - val_loss: 0.7788 - val_accuracy: 0.5000
Epoch 3/20
23/23 [==============================] - 2s 76ms/step - loss: 0.6467 - accuracy: 0.6630 - val_loss: 0.5531 - val_accuracy: 0.7391
Epoch 4/20
23/23 [==============================] - 2s 78ms/step - loss: 0.6145 - accuracy: 0.7500 - val_loss: 0.4644 - val_accuracy: 0.7609
Epoch 5/20
23/23 [==============================] - 2s 77ms/step - loss: 0.5012 - accuracy: 0.7772 - val_loss: 2.3875 - val_accuracy: 0.5000
Epoch 6/20
23/23 [==============================] - 2s 77ms/step - loss: 0.5692 - accuracy: 0.7500 - val_loss: 0.4663 - val_accuracy: 0.7391
Epoch 7/20
23/23 [==============================] - 2s 77ms/step - loss: 0.5228 - accuracy: 0.7609 - val_loss: 0.5116 - val_accuracy: 0.7609
Epoch 8/20
23/23 [==============================] - 2s 76ms/step - loss: 0.5238 - accuracy: 0.7554 - val_loss: 0.3976 - val_accuracy: 0.8043
Epoch 9/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4729 - accuracy: 0.7826 - val_loss: 0.4363 - val_accuracy: 0.8043
Epoch 10/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4605 - accuracy: 0.7717 - val_loss: 0.4103 - val_accuracy: 0.8043
Epoch 11/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4278 - accuracy: 0.7935 - val_loss: 0.4158 - val_accuracy: 0.8043
Epoch 12/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4115 - accuracy: 0.7826 - val_loss: 0.4217 - val_accuracy: 0.7826
Epoch 13/20
23/23 [==============================] - 2s 77ms/step - loss: 0.3899 - accuracy: 0.8315 - val_loss: 0.3926 - val_accuracy: 0.8478
Epoch 14/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4251 - accuracy: 0.8152 - val_loss: 0.3710 - val_accuracy: 0.8261
Epoch 15/20
23/23 [==============================] - 2s 77ms/step - loss: 0.3806 - accuracy: 0.8478 - val_loss: 0.4289 - val_accuracy: 0.8043
Epoch 16/20
23/23 [==============================] - 2s 75ms/step - loss: 0.4089 - accuracy: 0.8370 - val_loss: 0.4376 - val_accuracy: 0.8043
Epoch 17/20
23/23 [==============================] - 2s 77ms/step - loss: 0.3345 - accuracy: 0.8478 - val_loss: 0.4298 - val_accuracy: 0.8261
Epoch 18/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4262 - accuracy: 0.8424 - val_loss: 0.3981 - val_accuracy: 0.8478
Epoch 19/20
23/23 [==============================] - 2s 77ms/step - loss: 0.2890 - accuracy: 0.8804 - val_loss: 0.4454 - val_accuracy: 0.8043
Epoch 20/20
23/23 [==============================] - 2s 76ms/step - loss: 0.3629 - accuracy: 0.8533 - val_loss: 0.3811 - val_accuracy: 0.8696
```

Figure 25 COVID-19 vs pneumonia traning process

Similarly, for the task of classifying COVID-19 and pneumonia patients, the model also reaches a validation accuracy of about 85%. There is no big divergence between training and validation loss and accuracy which means the model does not overfit.

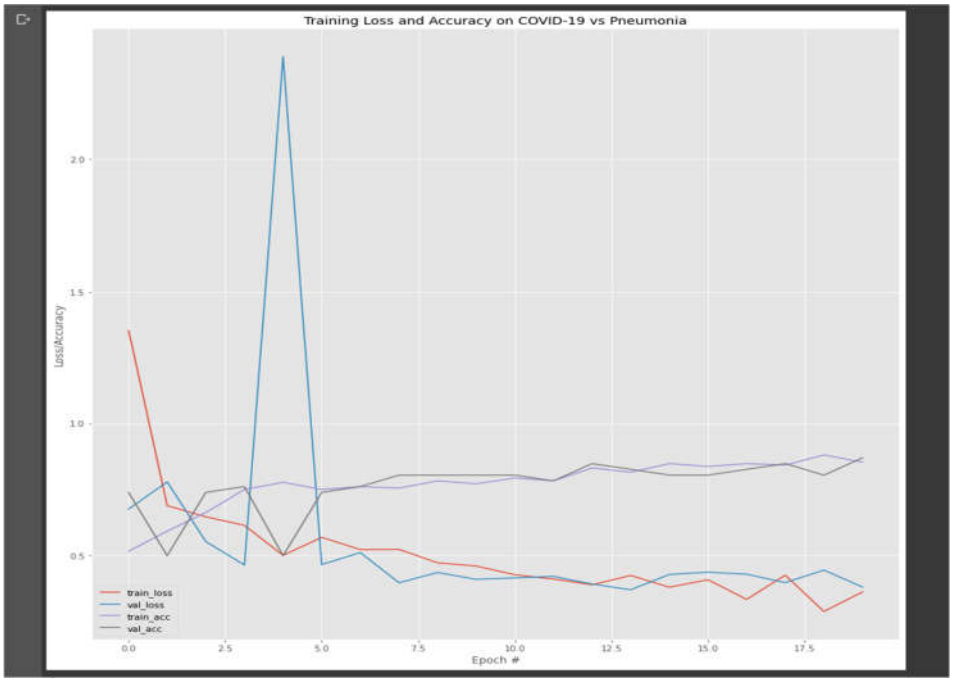

Figure 26 COVID-19 vs pneumonia training accuracy/lost history

```
Epoch 1/20
23/23 [==============================] - 2s 86ms/step - loss: 0.7399 - accuracy: 0.4674 - val_loss: 0.6852 - val_accuracy: 0.6522
Epoch 2/20
23/23 [==============================] - 2s 77ms/step - loss: 0.8000 - accuracy: 0.5652 - val_loss: 0.6370 - val_accuracy: 0.6304
Epoch 3/20
23/23 [==============================] - 2s 78ms/step - loss: 0.6697 - accuracy: 0.6902 - val_loss: 0.6070 - val_accuracy: 0.7391
Epoch 4/20
23/23 [==============================] - 2s 78ms/step - loss: 0.6464 - accuracy: 0.6957 - val_loss: 0.6327 - val_accuracy: 0.7391
Epoch 5/20
23/23 [==============================] - 2s 79ms/step - loss: 0.5888 - accuracy: 0.7391 - val_loss: 0.6135 - val_accuracy: 0.6957
Epoch 6/20
23/23 [==============================] - 2s 77ms/step - loss: 0.6203 - accuracy: 0.7283 - val_loss: 0.6325 - val_accuracy: 0.6522
Epoch 7/20
23/23 [==============================] - 2s 77ms/step - loss: 0.5719 - accuracy: 0.7391 - val_loss: 0.5946 - val_accuracy: 0.7391
Epoch 8/20
23/23 [==============================] - 2s 75ms/step - loss: 0.5070 - accuracy: 0.7609 - val_loss: 0.5825 - val_accuracy: 0.6957
Epoch 9/20
23/23 [==============================] - 2s 77ms/step - loss: 0.5512 - accuracy: 0.7391 - val_loss: 0.6833 - val_accuracy: 0.6957
Epoch 10/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4942 - accuracy: 0.7609 - val_loss: 0.7516 - val_accuracy: 0.6739
Epoch 11/20
23/23 [==============================] - 2s 78ms/step - loss: 0.5012 - accuracy: 0.7826 - val_loss: 0.8462 - val_accuracy: 0.6957
Epoch 12/20
23/23 [==============================] - 2s 75ms/step - loss: 0.5263 - accuracy: 0.7826 - val_loss: 0.6029 - val_accuracy: 0.7391
Epoch 13/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4899 - accuracy: 0.7989 - val_loss: 0.6064 - val_accuracy: 0.6739
Epoch 14/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4688 - accuracy: 0.8098 - val_loss: 0.5837 - val_accuracy: 0.6957
Epoch 15/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4382 - accuracy: 0.8261 - val_loss: 0.8432 - val_accuracy: 0.5870
Epoch 16/20
23/23 [==============================] - 2s 76ms/step - loss: 0.4834 - accuracy: 0.7989 - val_loss: 0.6282 - val_accuracy: 0.6522
Epoch 17/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4687 - accuracy: 0.8043 - val_loss: 0.5798 - val_accuracy: 0.6957
Epoch 18/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4219 - accuracy: 0.8478 - val_loss: 0.6475 - val_accuracy: 0.6957
Epoch 19/20
23/23 [==============================] - 2s 77ms/step - loss: 0.4273 - accuracy: 0.8207 - val_loss: 0.6864 - val_accuracy: 0.6957
Epoch 20/20
23/23 [==============================] - 2s 78ms/step - loss: 0.3776 - accuracy: 0.8587 - val_loss: 0.7435 - val_accuracy: 0.6304
```

Figure 27 COVID-19 vs non COVID-19 traning process

For the task of classifying COVID-19 positive and negative patients, the model only reaches a validation accuracy of about 75%. However, the model starts to overfit after about 5 epochs. More advanced deep learning techniques need to be used in order to achieve better results for this subproject.

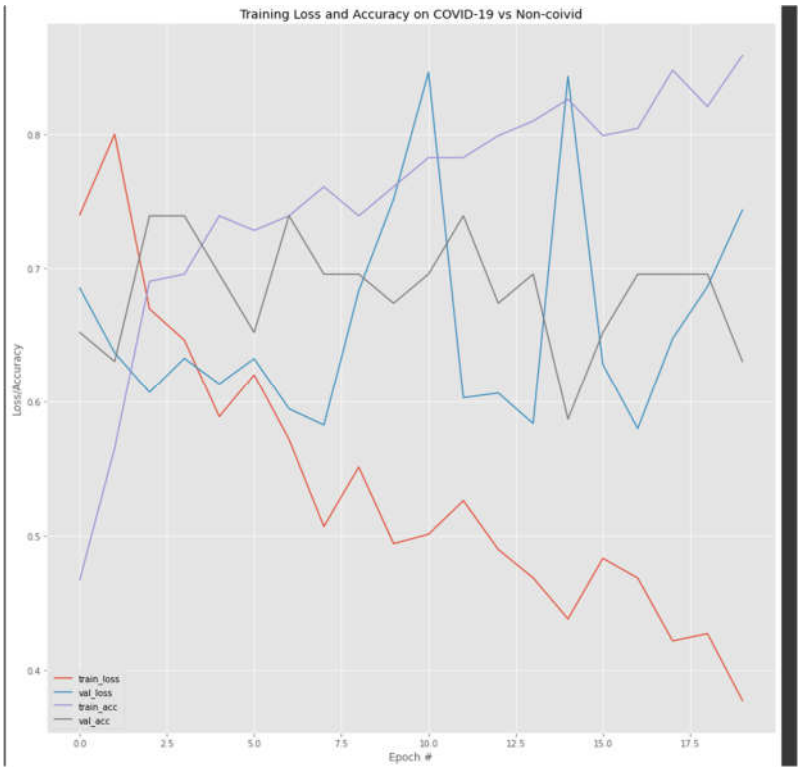

Figure 28 COVID-19 vs non COVID-19 training accuracy/lost history

6.2 Using a pretrained Convolutional Neural Network

A better approach would be taking advantage of a pretrained Convolutional Neural Network on a much bigger dataset. This project will use the architecture of the Xception networks [9] pretrained on the ImageNet dataset which includes more than 14 million images from over 20 thousand classes (for example cat, dog) [10]. Although the network was not trained on chest X-ray images, it is still useful for most computer vision tasks and can help this project achieve better results despite the lack of data. [7].

```
1  base_model = keras.applications.Xception(
2      weights='imagenet',  # Load weights pre-trained on ImageNet.
3      input_shape=(224, 224, 3),
4      include_top=False)  # Do not include the ImageNet classifier at the top.
5
6  # Freeze the base_model
7  base_model.trainable = False
8
9  # Create new model on top
10 inputs = keras.Input(shape=(224, 224, 3))
11
12 # The base model contains batchnorm layers. We want to keep them in inference mode
13 # when we unfreeze the base model for fine-tuning, so we make sure that the
14 # base_model is running in inference mode here.
15 x = base_model(inputs, training=False)
16 x = keras.layers.GlobalAveragePooling2D()(x)
17 x = keras.layers.Dropout(0.2)(x)  # Regularize with dropout
18 outputs = keras.layers.Dense(1, activation='sigmoid')(x)
19 model = keras.Model(inputs, outputs)
20
21 model.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
xception (Model)             (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

Figure 29 Using pretrained convnet

Line 1 from figure 29 instantiates the base model with loaded pretrained weights. Line 7 freezes all layers of the base model. Then a new small model is added on top of the

convolutional part (lines 10-18). Finally, this new model will be trained on our chest X-ray dataset. The trainable params number is now only about 2 thousand out of more than 20 million total params.

```
1  model.compile(optimizer=keras.optimizers.Adam(),
2                loss=keras.losses.BinaryCrossentropy(from_logits=True),
3                metrics=[keras.metrics.BinaryAccuracy()])
4
5  H = model.fit(
6    trainAug.flow(trainX, trainY, batch_size=BS),
7    steps_per_epoch=len(trainX) // BS,
8    validation_data=(testX, testY),
9    validation_steps=len(testX) // BS,
10   epochs=EPOCHS)
```

```
Epoch 1/20
23/23 [==============================] - 3s 140ms/step - loss: 0.6954 - binary_accuracy: 0.5163 - val_loss: 0.6845 - val_binary_accuracy: 0.5000
Epoch 2/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6761 - binary_accuracy: 0.5000 - val_loss: 0.6682 - val_binary_accuracy: 0.5217
Epoch 3/20
23/23 [==============================] - 2s 102ms/step - loss: 0.6657 - binary_accuracy: 0.5652 - val_loss: 0.6547 - val_binary_accuracy: 0.6304
Epoch 4/20
23/23 [==============================] - 2s 102ms/step - loss: 0.6456 - binary_accuracy: 0.7446 - val_loss: 0.6452 - val_binary_accuracy: 0.6739
Epoch 5/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6370 - binary_accuracy: 0.7120 - val_loss: 0.6383 - val_binary_accuracy: 0.7174
Epoch 6/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6310 - binary_accuracy: 0.7772 - val_loss: 0.6309 - val_binary_accuracy: 0.7391
Epoch 7/20
23/23 [==============================] - 2s 103ms/step - loss: 0.6270 - binary_accuracy: 0.7500 - val_loss: 0.6276 - val_binary_accuracy: 0.7391
Epoch 8/20
23/23 [==============================] - 2s 105ms/step - loss: 0.6138 - binary_accuracy: 0.7935 - val_loss: 0.6235 - val_binary_accuracy: 0.7391
Epoch 9/20
23/23 [==============================] - 2s 103ms/step - loss: 0.6141 - binary_accuracy: 0.8043 - val_loss: 0.6205 - val_binary_accuracy: 0.7391
Epoch 10/20
23/23 [==============================] - 2s 103ms/step - loss: 0.6178 - binary_accuracy: 0.7935 - val_loss: 0.6186 - val_binary_accuracy: 0.7391
Epoch 11/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6108 - binary_accuracy: 0.7880 - val_loss: 0.6155 - val_binary_accuracy: 0.7609
Epoch 12/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6021 - binary_accuracy: 0.8261 - val_loss: 0.6140 - val_binary_accuracy: 0.7609
Epoch 13/20
23/23 [==============================] - 2s 104ms/step - loss: 0.6088 - binary_accuracy: 0.8207 - val_loss: 0.6135 - val_binary_accuracy: 0.7391
Epoch 14/20
23/23 [==============================] - 2s 102ms/step - loss: 0.6030 - binary_accuracy: 0.7880 - val_loss: 0.6115 - val_binary_accuracy: 0.7826
Epoch 15/20
23/23 [==============================] - 2s 102ms/step - loss: 0.6034 - binary_accuracy: 0.8424 - val_loss: 0.6083 - val_binary_accuracy: 0.7609
Epoch 16/20
23/23 [==============================] - 2s 103ms/step - loss: 0.5937 - binary_accuracy: 0.8207 - val_loss: 0.6066 - val_binary_accuracy: 0.7609
Epoch 17/20
23/23 [==============================] - 2s 105ms/step - loss: 0.5902 - binary_accuracy: 0.8641 - val_loss: 0.6060 - val_binary_accuracy: 0.7609
Epoch 18/20
23/23 [==============================] - 2s 104ms/step - loss: 0.5865 - binary_accuracy: 0.8696 - val_loss: 0.6054 - val_binary_accuracy: 0.7826
Epoch 19/20
23/23 [==============================] - 2s 106ms/step - loss: 0.5945 - binary_accuracy: 0.8424 - val_loss: 0.6028 - val_binary_accuracy: 0.7826
Epoch 20/20
23/23 [==============================] - 2s 105ms/step - loss: 0.5855 - binary_accuracy: 0.8587 - val_loss: 0.6028 - val_binary_accuracy: 0.7826
```

Figure 30 COVID-19 vs Normal training process

The model trains quickly thanks to its small size. After 20 epochs, it could not break 80% validation accuracy, which is worse than the baseline in chapter 6.1. However, from figure 30, the model does not overfit and can potentially achieve higher accuracy training with more epochs.

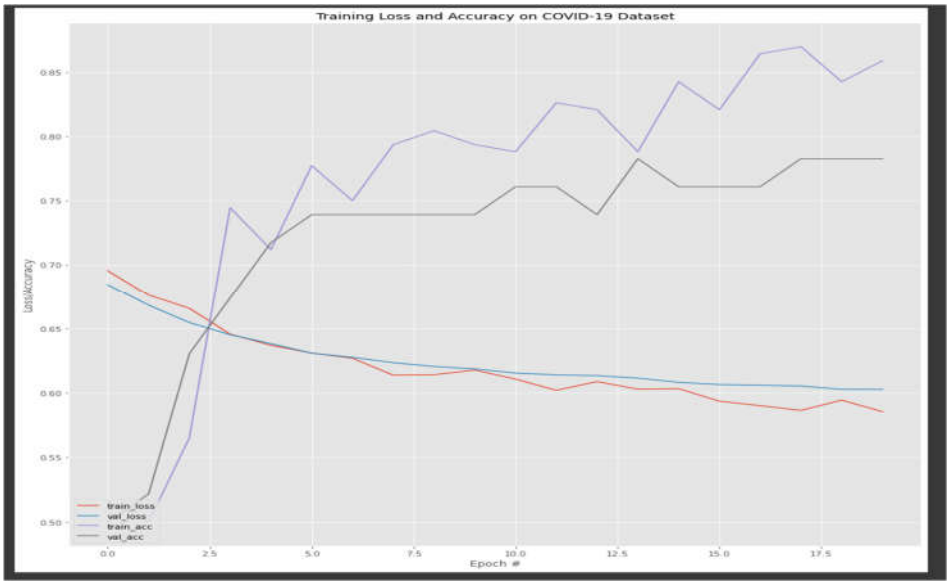

Figure 31 COVID-19 vs normal training accuracy/lost history

The model reaches a validation accuracy of about 85% in the task of classifying COVID-19 and pneumonia patients, which is like the baseline model. The model is also not over-fitting.

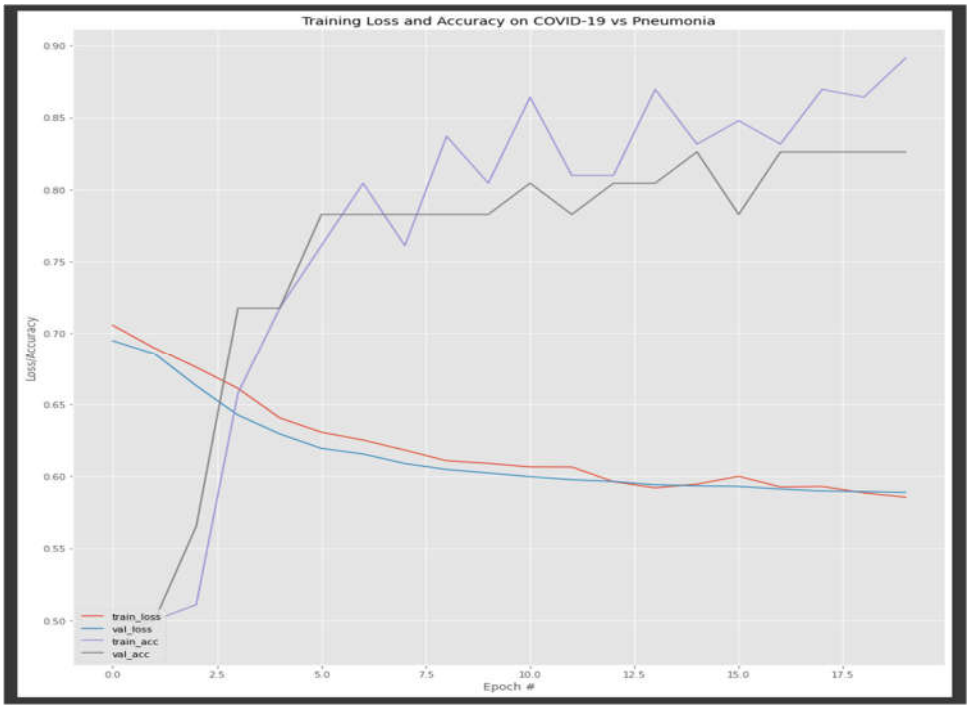

Figure 32 COVID-19 vs pneumonia training accuracy/lost history

For the task of classifying COVID-19 positive and negative patients, the model reaches a validation accuracy of about 80% which is better than the baseline model of 75%.

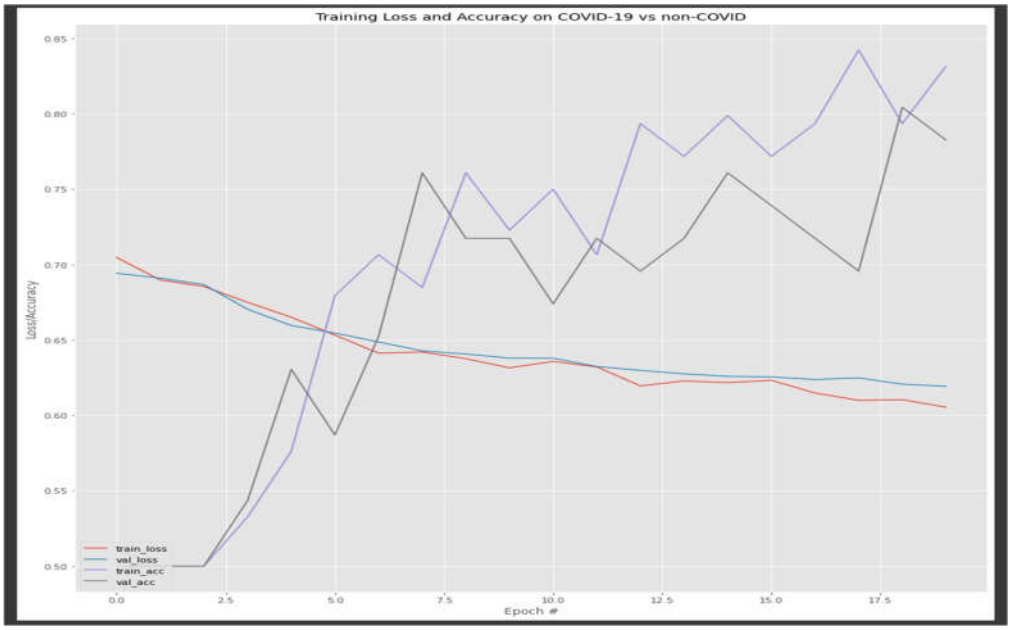

Figure 33 COVID-19 vs non-COVID training accuracy/lost history

6.3 Fine-tuning a pretrained Convolutional Neural Network

The last optional step to boost the performance of the models is unfreezing the base model (line 6, figure 34) and training the entire model end-to-end for a few more epochs with a low learning rate (lines 9-11).

```
6  base_model.trainable = True
7  model.summary()
8
9  model.compile(optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate
10                loss=keras.losses.BinaryCrossentropy(from_logits=True),
11                metrics=[keras.metrics.BinaryAccuracy()])
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
xception (Model)             (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 20,809,001
Non-trainable params: 54,528
_________________________________________________________________
```

Figure 34 Fine-tuning a pretrained network

After 10 more epochs, fine-tuning helps the model reach more than 93% validation accuracy in classifying COVID-19 patients and healthy people, which is a huge improvement over the previous result.

```
1  H = model.fit(
2    trainAug.flow(trainX, trainY, batch_size=BS),
3    steps_per_epoch=len(trainX) // BS,
4    validation_data=(testX, testY),
5    validation_steps=len(testX) // BS,
6    epochs=10)
```

```
Epoch 1/10
23/23 [==============================] - 4s 177ms/step - loss: 0.5999 - binary_accuracy: 0.7989 - val_loss: 0.5951 - val_binary_accuracy: 0.7391
Epoch 2/10
23/23 [==============================] - 3s 140ms/step - loss: 0.5686 - binary_accuracy: 0.8913 - val_loss: 0.5937 - val_binary_accuracy: 0.7391
Epoch 3/10
23/23 [==============================] - 3s 140ms/step - loss: 0.5466 - binary_accuracy: 0.9185 - val_loss: 0.5629 - val_binary_accuracy: 0.8696
Epoch 4/10
23/23 [==============================] - 3s 138ms/step - loss: 0.5316 - binary_accuracy: 0.9457 - val_loss: 0.5854 - val_binary_accuracy: 0.7391
Epoch 5/10
23/23 [==============================] - 3s 139ms/step - loss: 0.5207 - binary_accuracy: 0.9783 - val_loss: 0.5625 - val_binary_accuracy: 0.8261
Epoch 6/10
23/23 [==============================] - 3s 140ms/step - loss: 0.5167 - binary_accuracy: 0.9837 - val_loss: 0.5354 - val_binary_accuracy: 0.9565
Epoch 7/10
23/23 [==============================] - 3s 139ms/step - loss: 0.5107 - binary_accuracy: 0.9946 - val_loss: 0.5402 - val_binary_accuracy: 0.9130
Epoch 8/10
23/23 [==============================] - 3s 140ms/step - loss: 0.5078 - binary_accuracy: 0.9946 - val_loss: 0.5362 - val_binary_accuracy: 0.9348
Epoch 9/10
23/23 [==============================] - 3s 140ms/step - loss: 0.5069 - binary_accuracy: 0.9946 - val_loss: 0.5421 - val_binary_accuracy: 0.9130
Epoch 10/10
23/23 [==============================] - 3s 138ms/step - loss: 0.5061 - binary_accuracy: 0.9946 - val_loss: 0.5321 - val_binary_accuracy: 0.9348
```

Figure 35 Fine-tuning training process

From figure 36, there is no big divergence between training and validation loss and accuracy which means the model does not suffer heavily from overfitting.

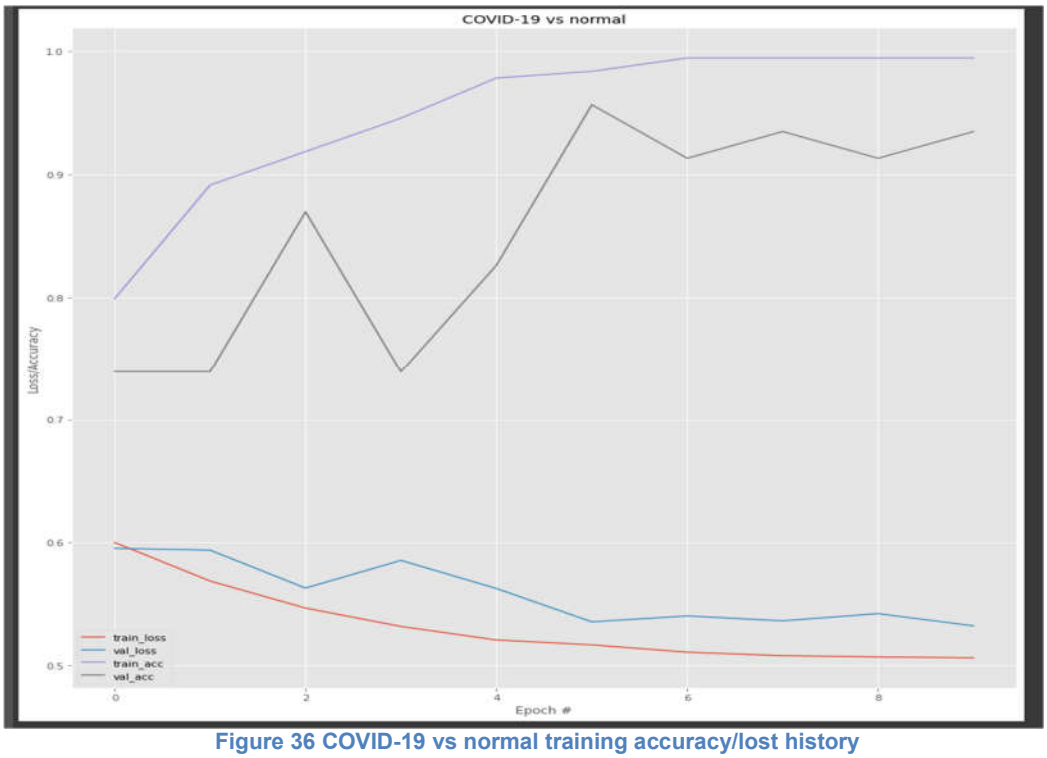

Figure 36 COVID-19 vs normal training accuracy/lost history

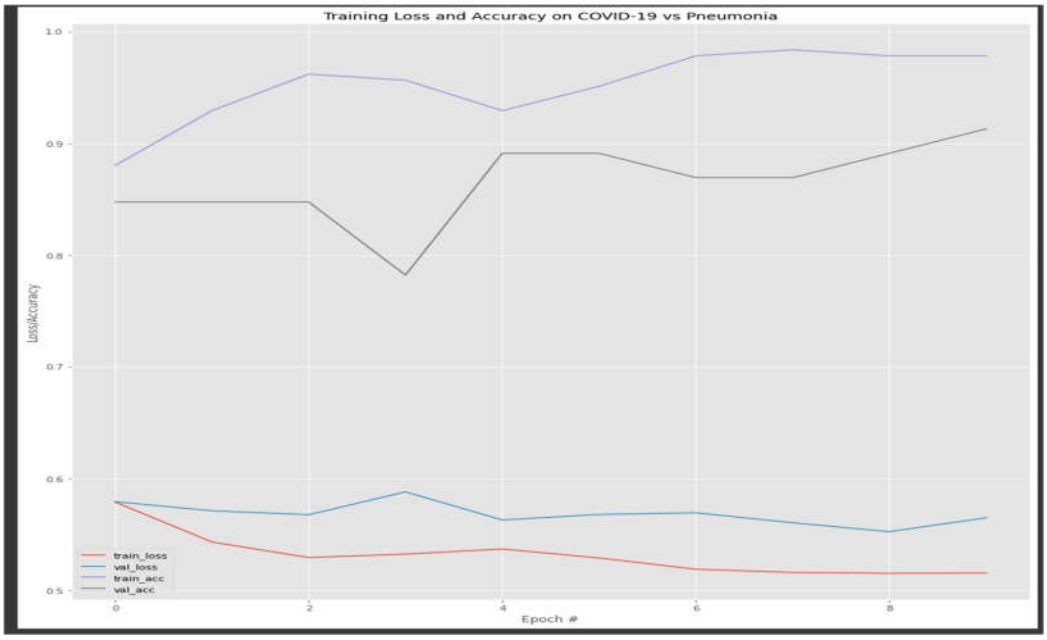

Figure 37 COVID-19 vs pneumonia training accuracy/lost history

From figure 37, in the task of classifying COVID-19 and pneumonia patients, fine-tuning also helps the model gain over 5% and get to 90% accuracy.

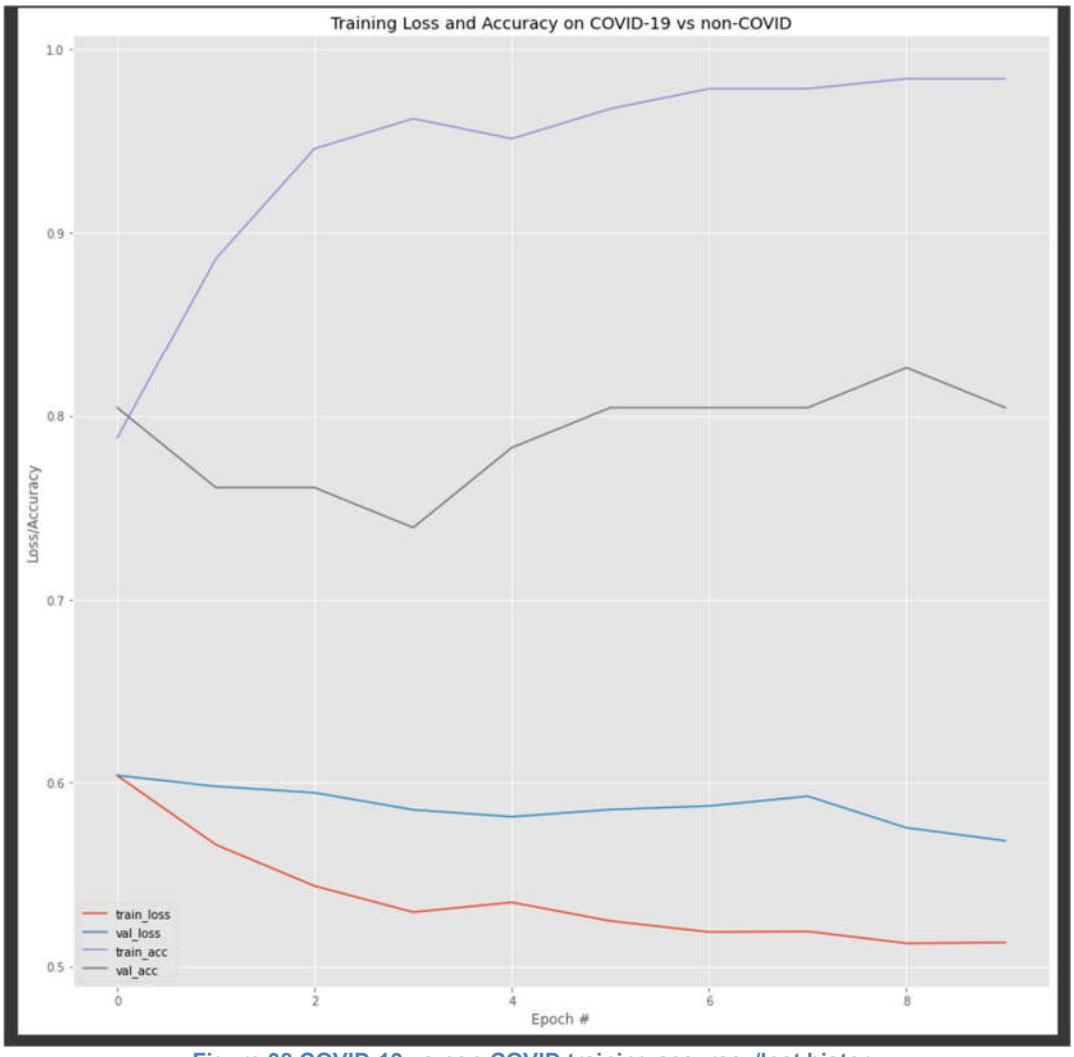

Figure 38 COVID-19 vs non-COVID training accuracy/lost history

Finally, for the task of classifying COVID-19 positive and negative patients, fine-tuning only brings a small improvement in the model.

7 Result

To evaluate the model, the predictions on the evaluation dataset need to be made, which then will be used to generate the prediction indices. The classification report is generated by calling sklearn's method [5].

```
[ ]  1  from sklearn.metrics import classification_report
     2
     3  predIdxs = model.predict(testX, batch_size=BS)
     4  predIdxs = predIdxs > 0.5
     5  predIdxs = 1*predIdxs
     6
     7  print(classification_report(testY, predIdxs,
     8    target_names=lb.classes_))
```

```
              precision    recall  f1-score   support

       covid       0.92      0.96      0.94        23
      normal       0.95      0.91      0.93        23

    accuracy                           0.93        46
   macro avg       0.94      0.93      0.93        46
weighted avg       0.94      0.93      0.93        46
```

```
[ ]  1  from sklearn.metrics import confusion_matrix
     2
     3  cm = confusion_matrix(testY, predIdxs)
     4  total = sum(sum(cm))
     5  acc = (cm[0, 0] + cm[1, 1]) / total
     6  sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
     7  specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
     8
     9  # show the confusion matrix, accuracy, sensitivity, and specificity
    10  print(cm)
    11  print("acc: {:.4f}".format(acc))
    12  print("sensitivity: {:.4f}".format(sensitivity))
    13  print("specificity: {:.4f}".format(specificity))
```

```
[[22  1]
 [ 2 21]]
acc: 0.9348
sensitivity: 0.9565
specificity: 0.9130
```

Figure 39 COVID-19 vs Normal evaluation

Line 3 in the second script generates the confusion matrix. From it, the accuracy, sensitivity, and specificity metrics are derived. The model achieves 93% accuracy on classifying COVID-19 infected and healthy patients. The sensitivity metric of 96% means that out of 100 patients who do have COVID-19, the model accurately identifies 96 patients. The specificity metric of 91% means that out of 100 patients who do not have COVID-19, the model accurately identifies 91 patients and misclassifies 9 patients.

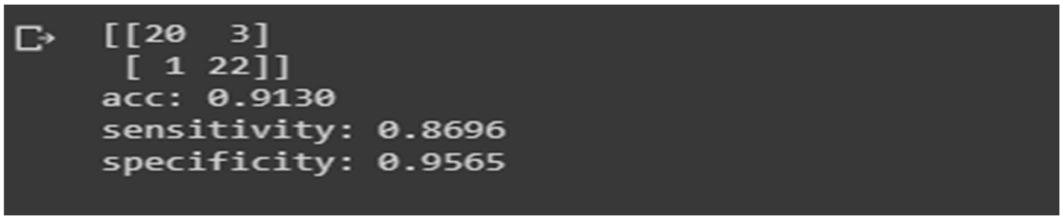

Figure 40 COVID-19 vs Pneumonia evaluation

Figure 40 shows the confusion matrix and the metrics of the model on classifying COVID-19 and Pneumonia patients. The model achieves an overall 91% accuracy on this task. The sensitivity metric of 87% means that out of 100 patients who do have COVID-19, the model accurately identifies 87 patients and misses out 13 patients. The specificity metric of 96% means that out of 100 patients who have pneumonia but not COVID-19, the model accurately identifies 96 patients and only misclassifies 4 patients.

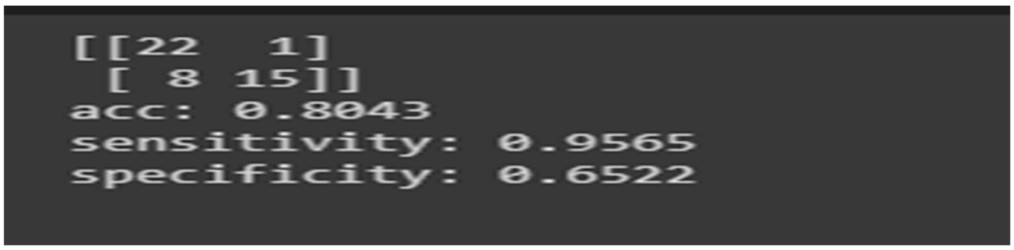

Figure 41 COVID-19 vs Non COVID-19 evaluation

Figure 41 shows the confusion matrix and the metrics of the model on classifying COVID-19 positive and negative patients. The model achieves an overall 80% accuracy on this task. The sensitivity metric of 96% means that out of 100 patients who do have COVID-19, the model accurately identifies 96 patients and misses out 4 patients. The specificity metric of 65% means that out of 100 patients who do not have COVID-19, the model accurately identifies 65 patients and misclassifies 35 patients as COVID-19 positive.

8 Conclusion

This project includes setting up a cloud-based working environment with Google Colab, collecting the datasets, training Deep Learning models, and evaluating the models. Using modern Deep Learning techniques such as using pre-trained networks and fine-tuning as well as regularizations such as data augmentation and dropout to fight overfitting, our model achieves an overall accuracy of 93% on the most realistic task of detecting COVID-19 patients among healthy normal people. Out of 100 patients who do have COVID-19, the model accurately identifies 96 patients and misses out 4 patients. These 4 patients, after being classified as COVID-19 negative, might infect other people. Out of 100 normal patients who do not have COVID-19, the model accurately identifies 91 patients as healthy and misclassifies 9 patients as COVID-19 positive. Overall, the model did a decent job as a COVID-19 detector despite being trained on a dataset of only 115 images for each class.

The main limitation of this project is the lack of data, more specifically chest X-Ray images of COVID-19 infected patients. With this amount of data available, the result of this project is simply not reliable enough. Hopefully in the future the COVID-19 dataset will grow larger.

This project still has a lot of room for improvement. First, the current model only uses image data. Future models can take different data types into use such as patients' health info, age, gender. Second, this project only uses basic Deep Learning techniques, and models are trained in a very short time. Using more advanced methods and training the models in a longer time could achieve better results. Finally, visualizing the opacities in the chest X-Ray images can bring a better understanding of how an image is classified.

References

1. Hinton GE, Osindero S, Teh YW. A Fast Learning Algorithm for Deep Belief Nets. Neural Computation. 2006;18(7):1527-1554.
   URL: https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf. Accessed 19 November 2018

2. Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Neural Information Processing Systems Conference; 2012. Available from: Google Scholar.
   URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. Accessed 19 November 2018

3. Cohen et al. COVID-19 image data collection.
   URL: https://github.com/ieee8023/covid-chestxray-dataset. Accessed 13 April 2020.

4. Chollet F. Deep Learning with Python; 2017. Manning Publications.

5. Rosebrock A. Detecting COVID-19 in X-ray images with Keras, TensorFlow, and Deep Learning.
   URL: https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning

6. Radiological Society of North America. RSNA pneumonia detection challenge.
   URL: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge. Accessed 14 April 2020.

7. Chollet F. Building powerful image classification models using very little data
   URL: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html . Accessed 18 April 2020.

8. How to use Kaggle. URL: https://www.kaggle.com/docs/api Accessed 27 April 2020.

9. Zahavi G. What are lung opacities?
   URL: https://www.kaggle.com/zahaviguy/what-are-lung-opacities Accessed 5 May 2020.

10. Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions. 2017.
    URL: https://arxiv.org/abs/1610.02357. Accessed 22 May 2020

11. ImageNet. URL: http://image-net.org/about-stats Accessed 22 May 2020.